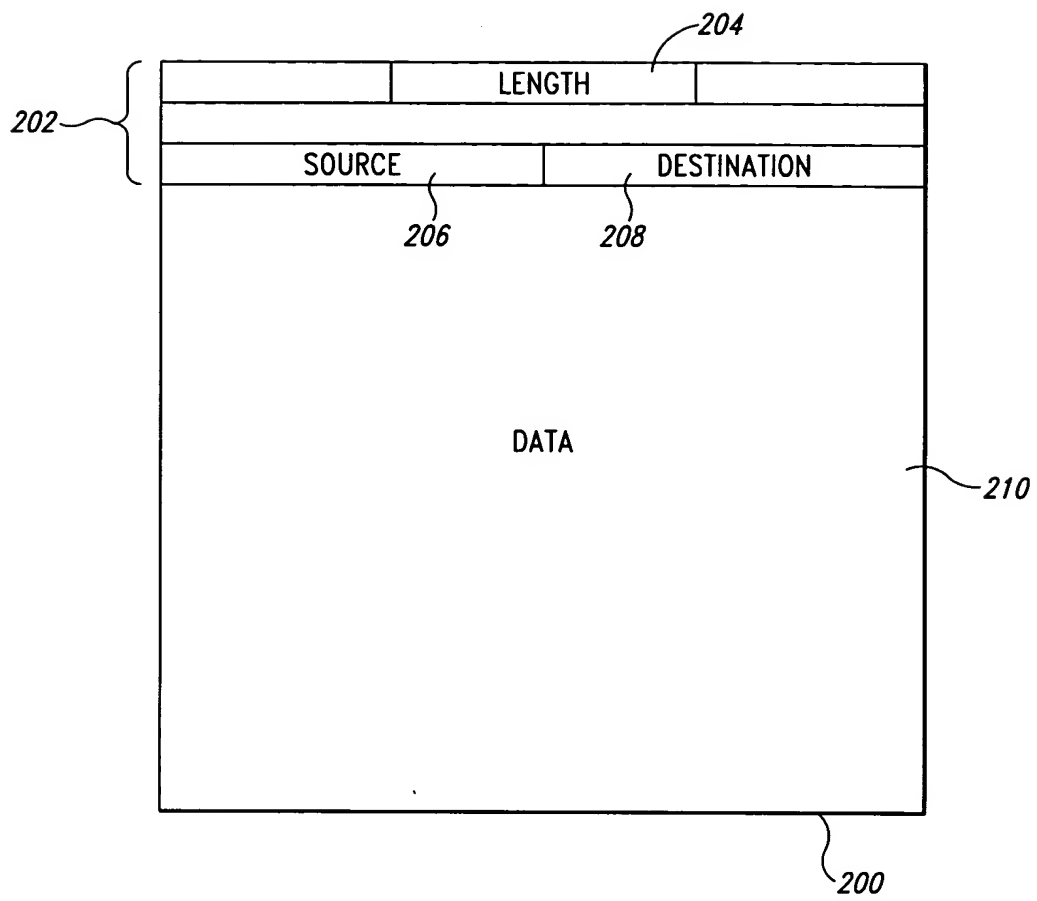
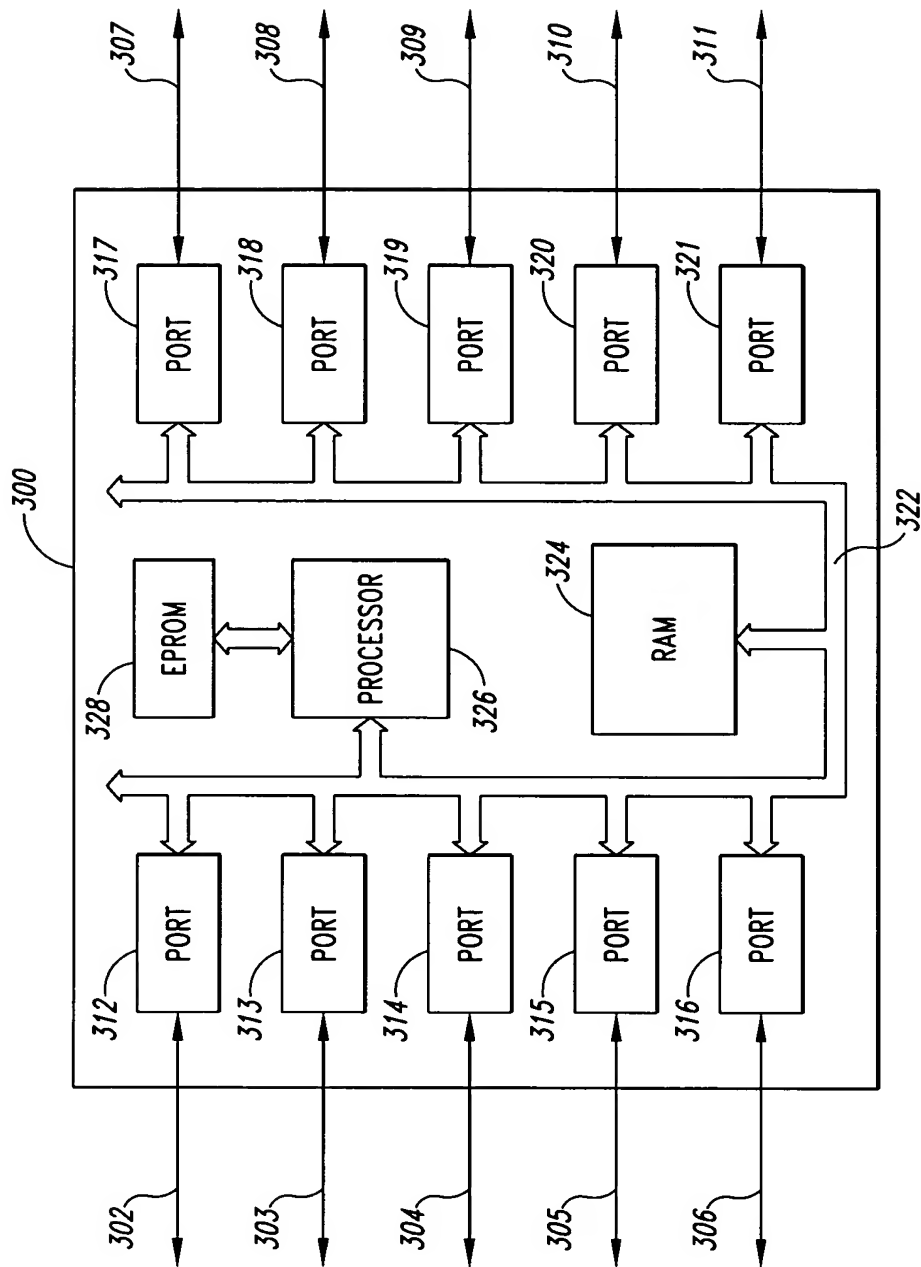
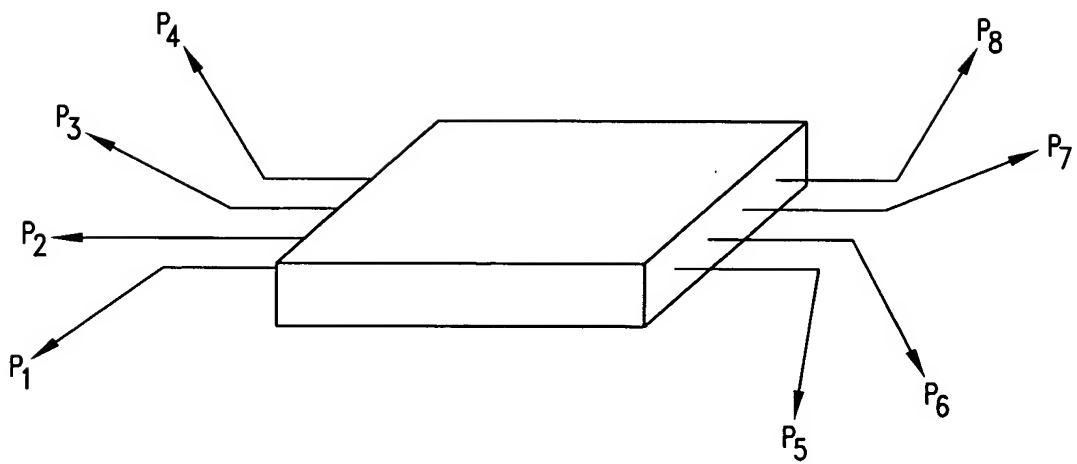


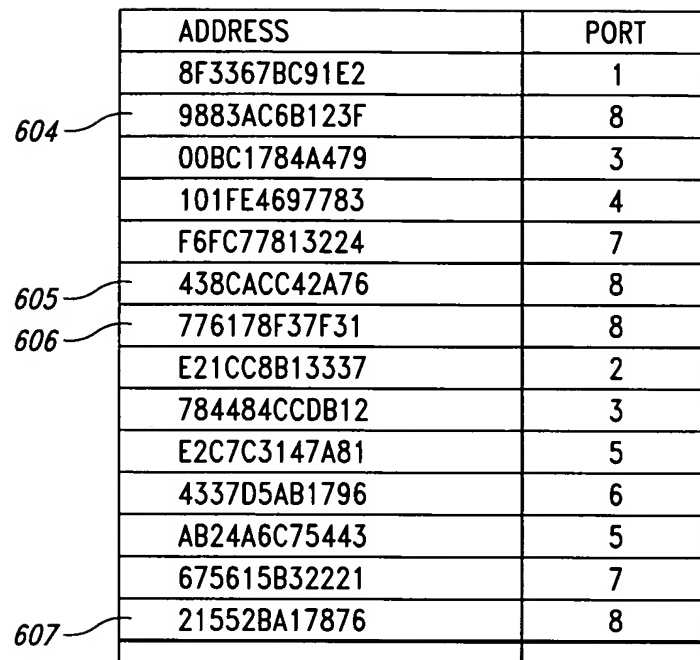
Fig. 1

*Fig. 2*

*Fig. 3*

ADDRESS	PORT
NODE A	X
PRINTER	X
NODE B	Y
NODE C	Y

Fig. 4*Fig. 5*



The diagram shows a table with two columns: ADDRESS and PORT. Callout 602 points to the top right corner of the table. Callout 604 points to the first three rows of the table. Callout 605 points to the row with address 438CACC42A76. Callout 606 points to the row with address 776178F37F31. Callout 607 points to the last row of the table.

ADDRESS	PORT
8F3367BC91E2	1
9883AC6B123F	8
00BC1784A479	3
101FE4697783	4
F6FC77813224	7
438CACC42A76	8
776178F37F31	8
E21CC8B13337	2
784484CCDB12	3
E2C7C3147A81	5
4337D5AB1796	6
AB24A6C75443	5
675615B32221	7
21552BA17876	8

Fig. 6

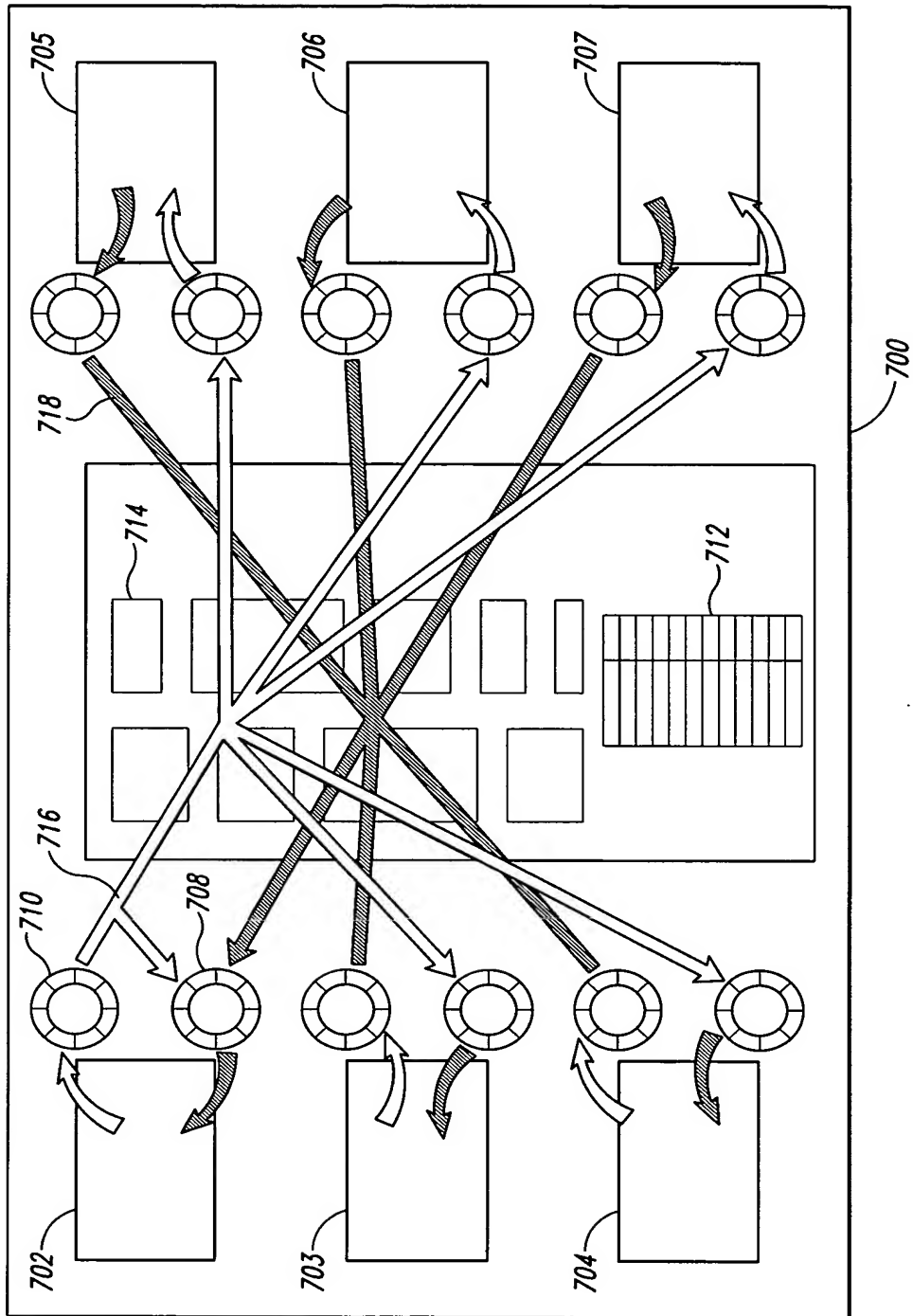
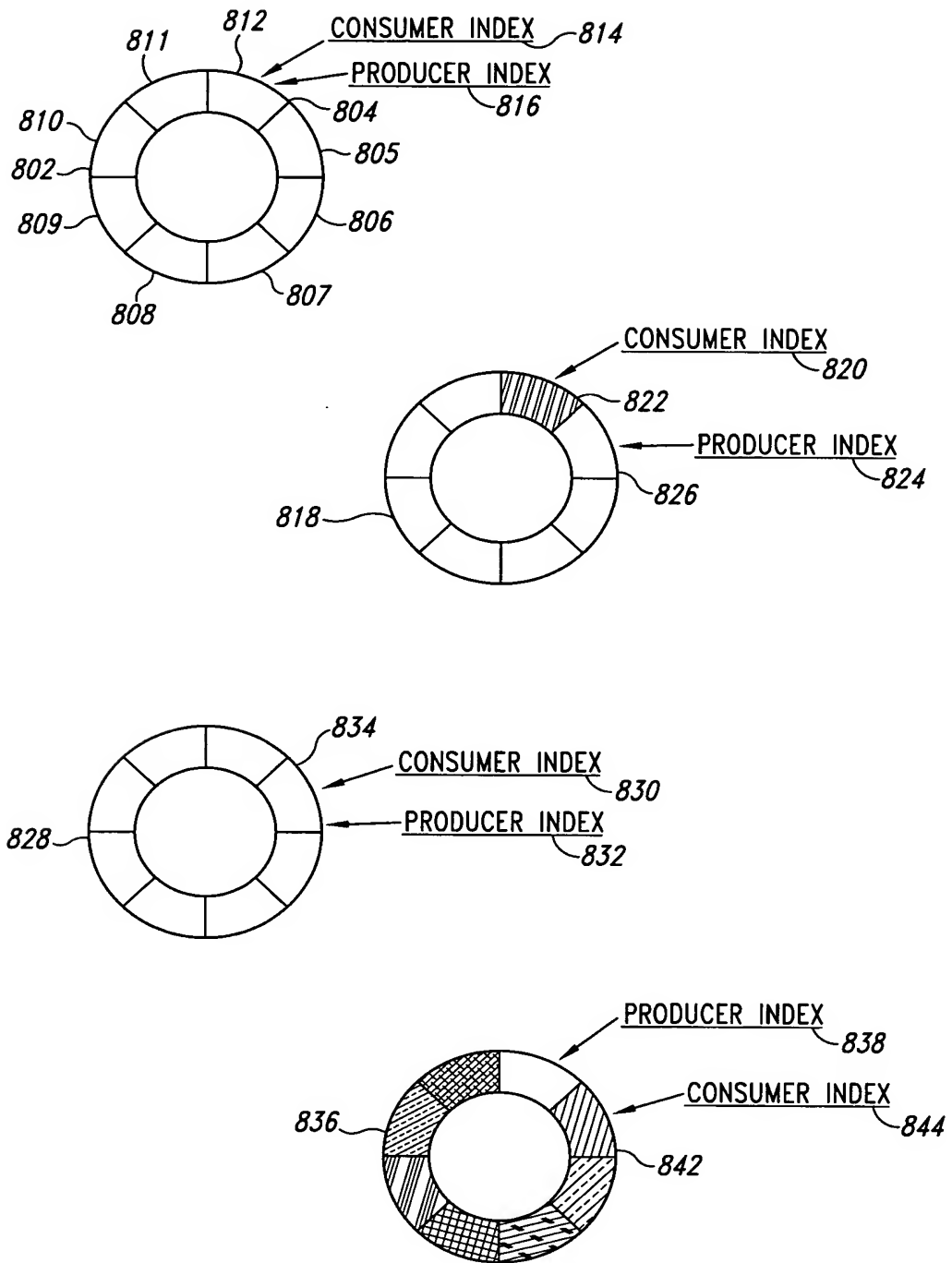
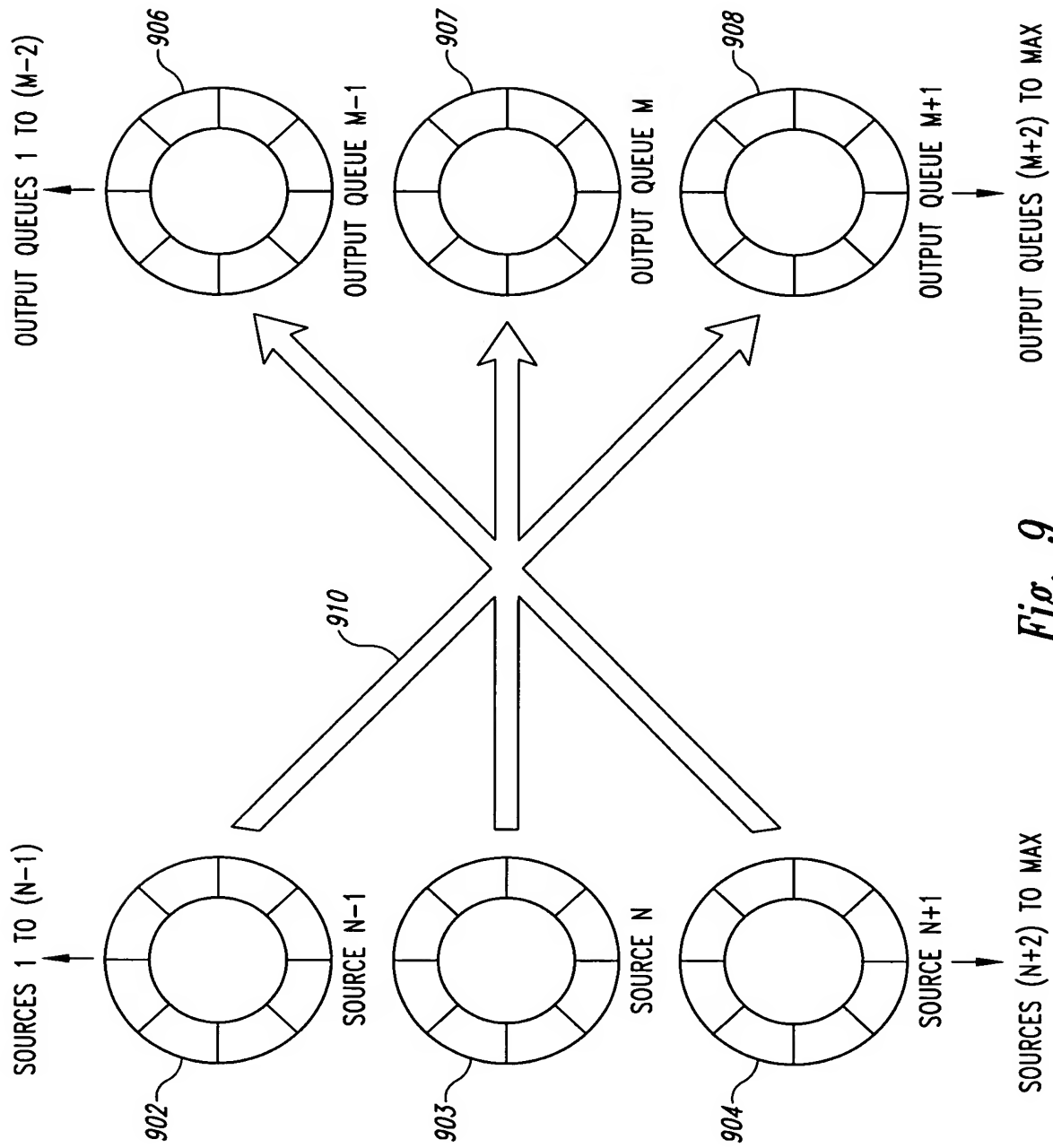


Fig. 7

*Fig. 8*

*Fig. 9*

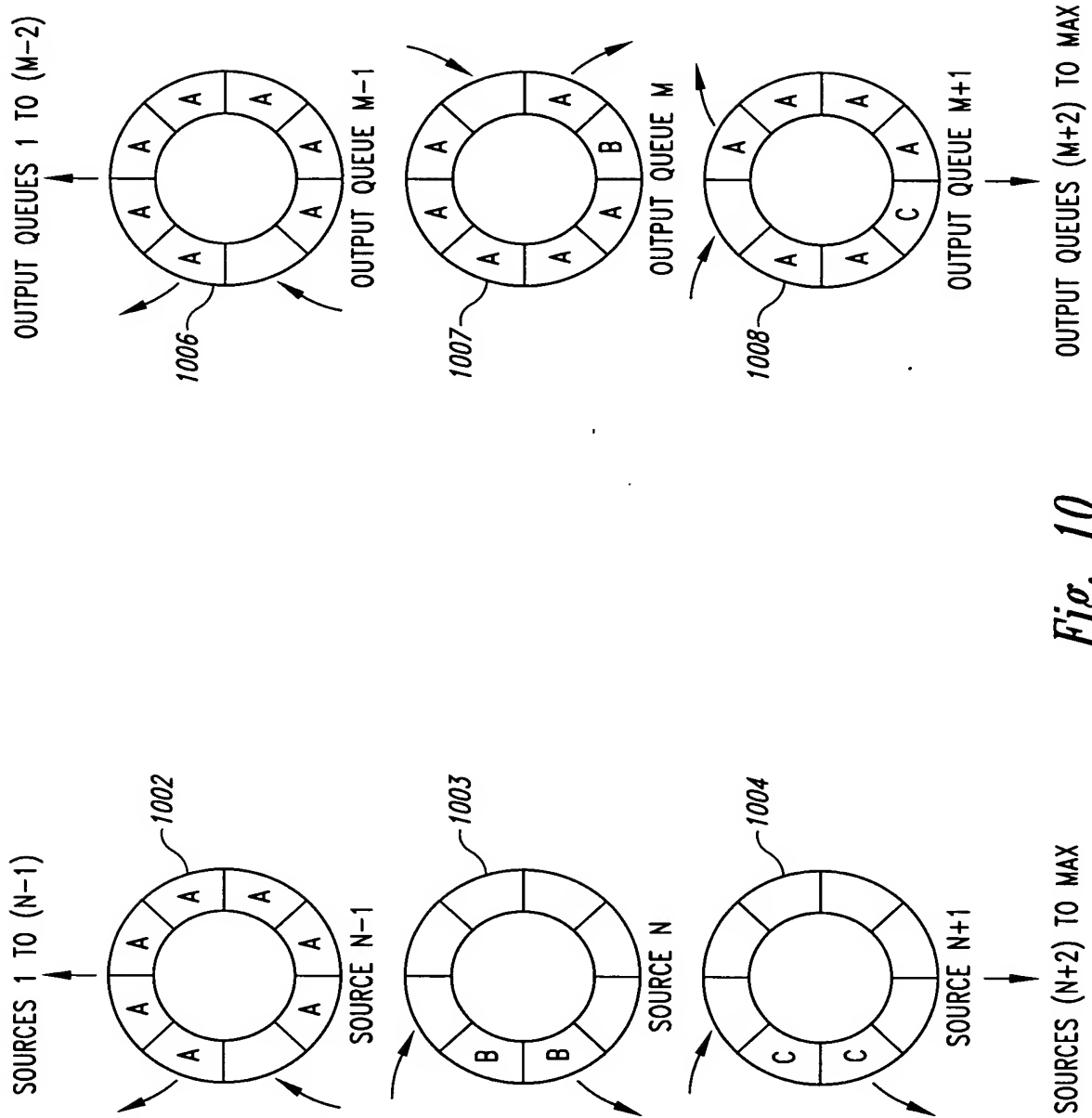


Fig. 10

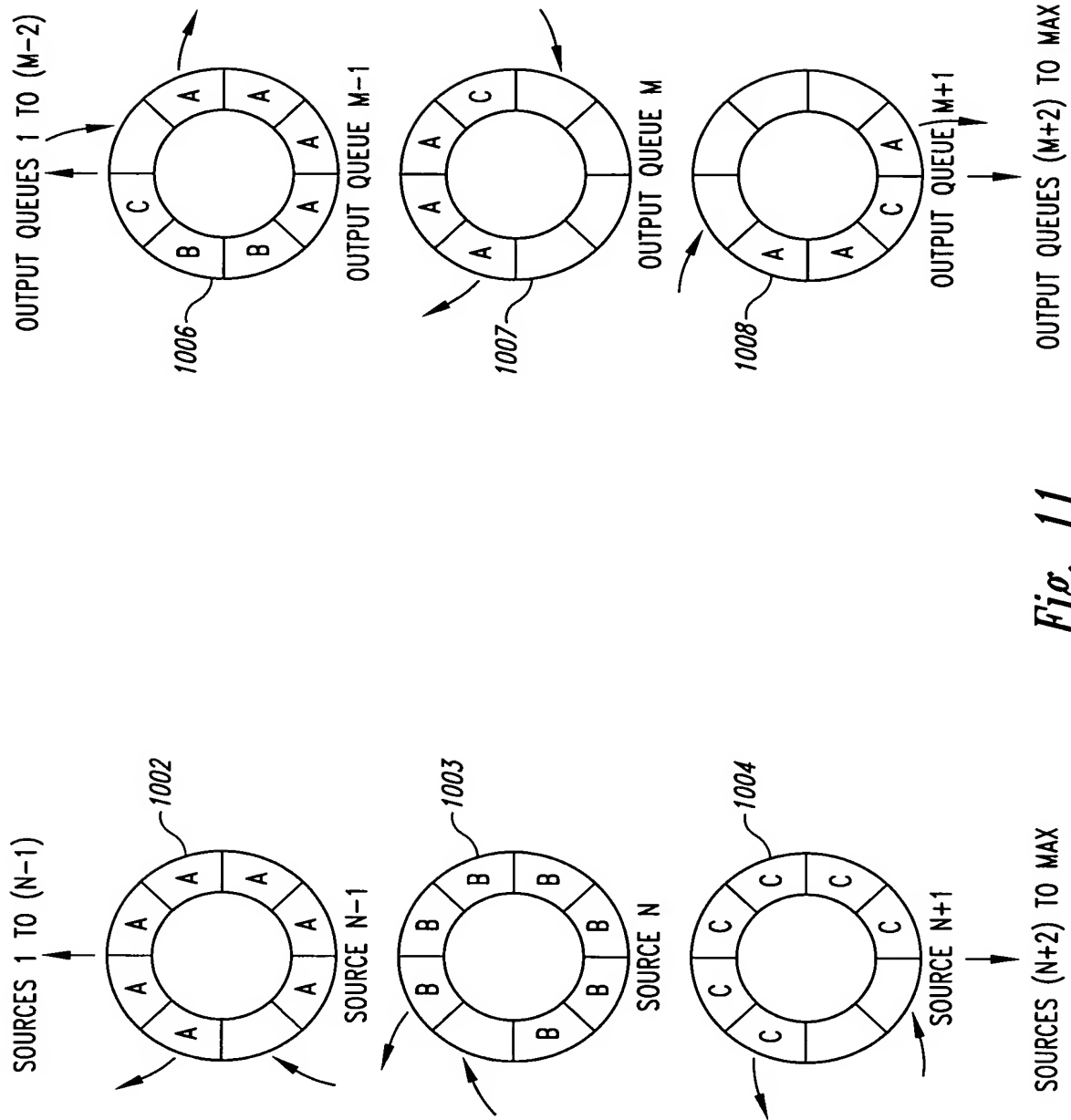


Fig. 11

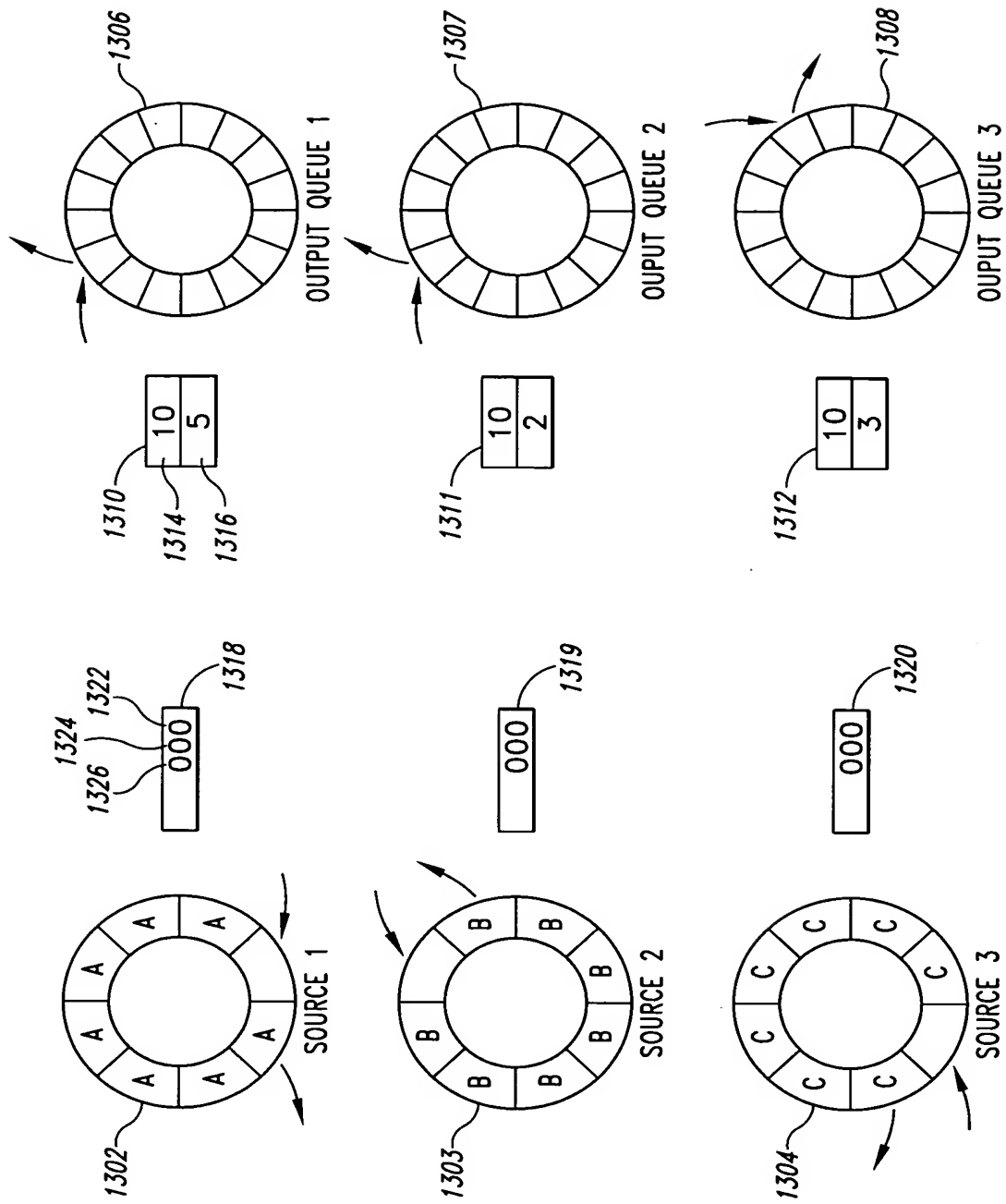


Fig. 13A

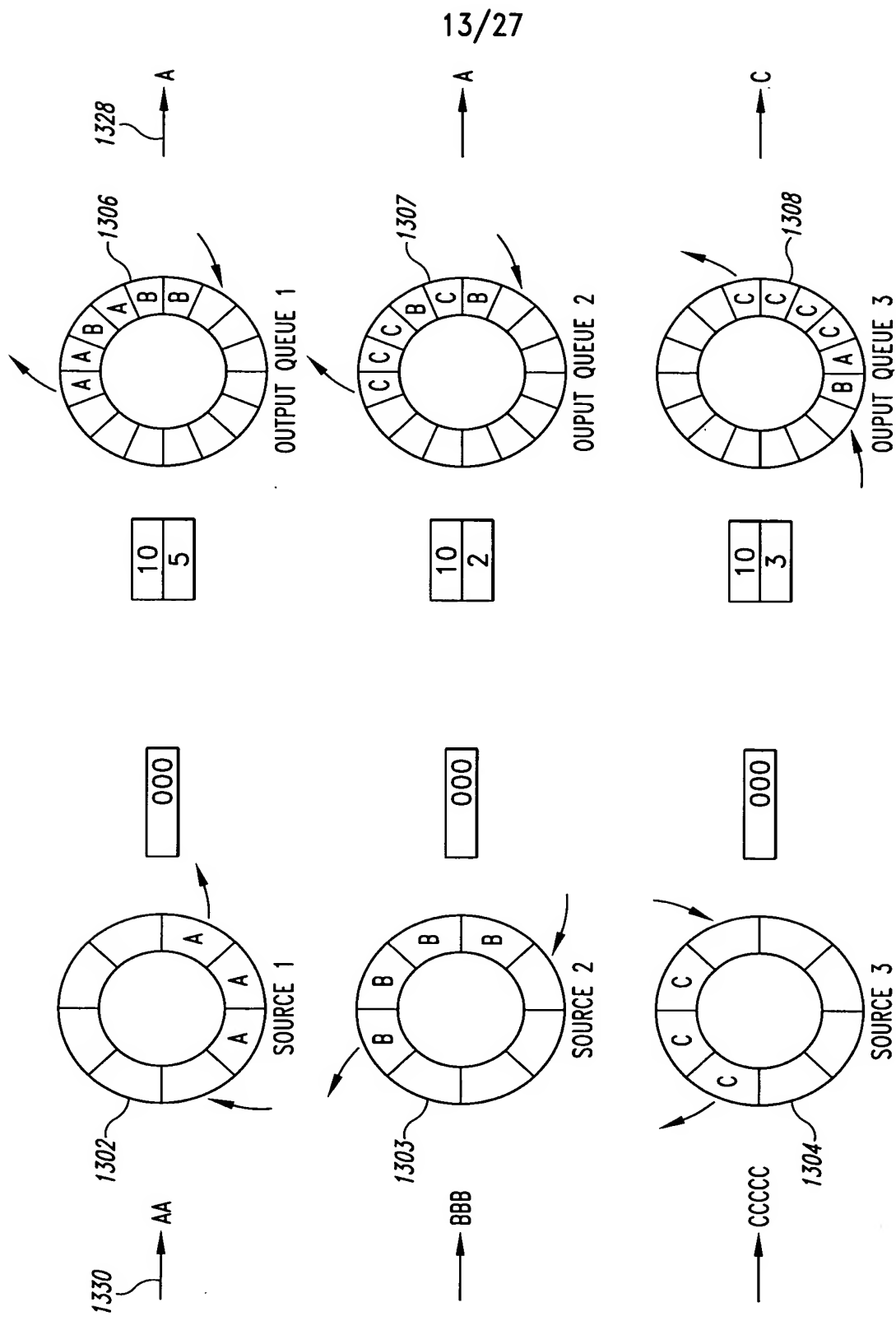


Fig. 13B

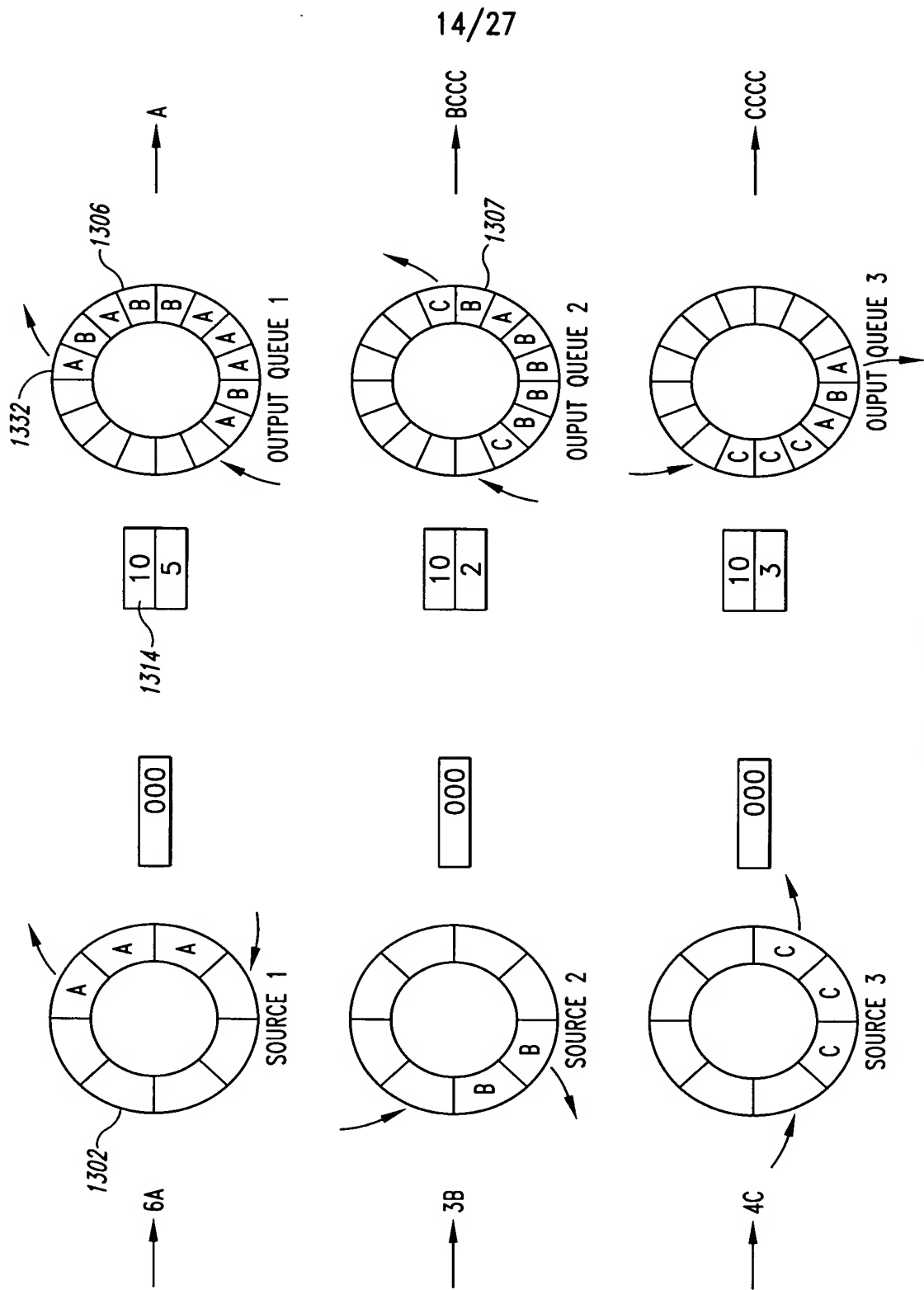


Fig. 13C

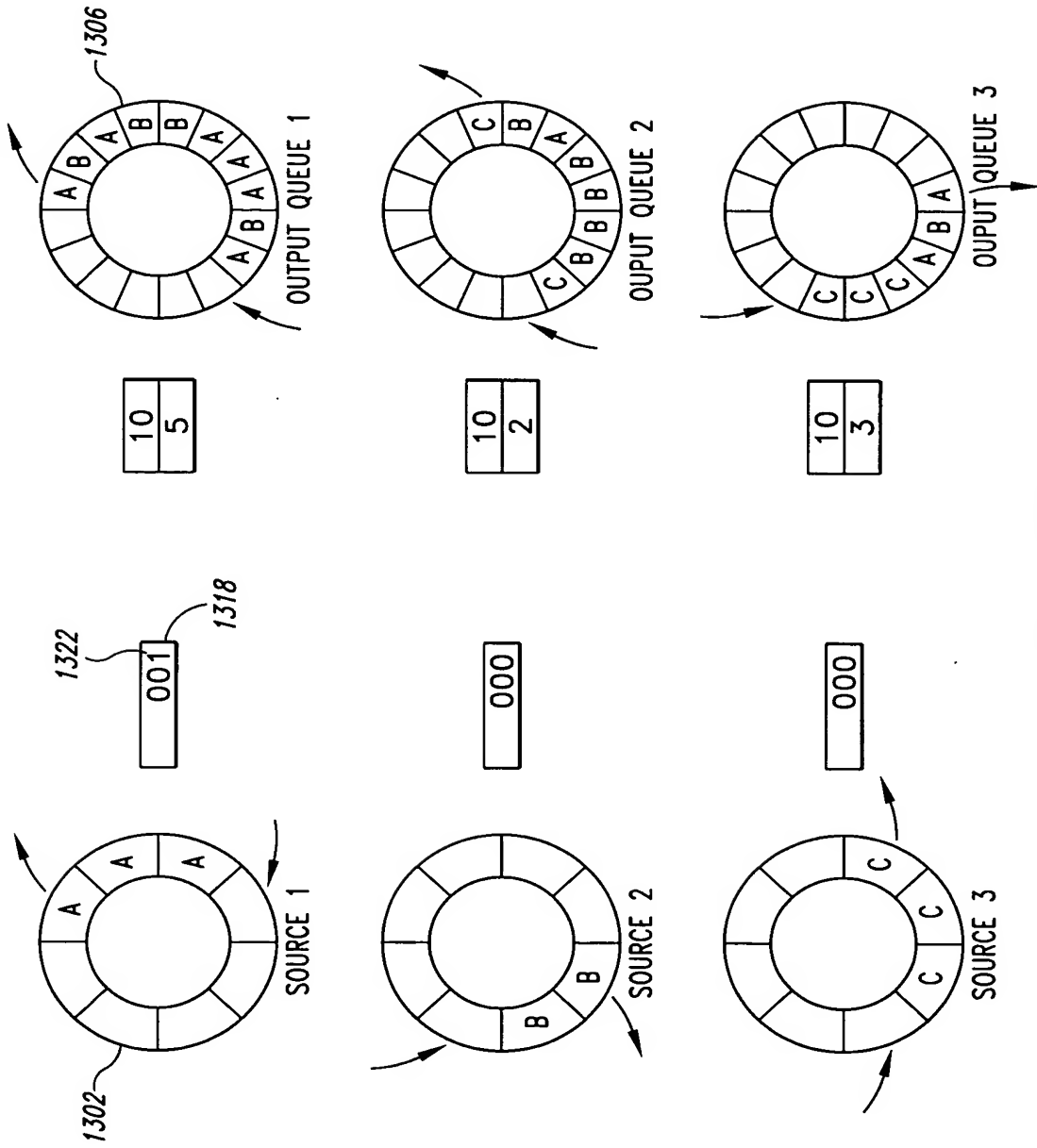


Fig. 13D

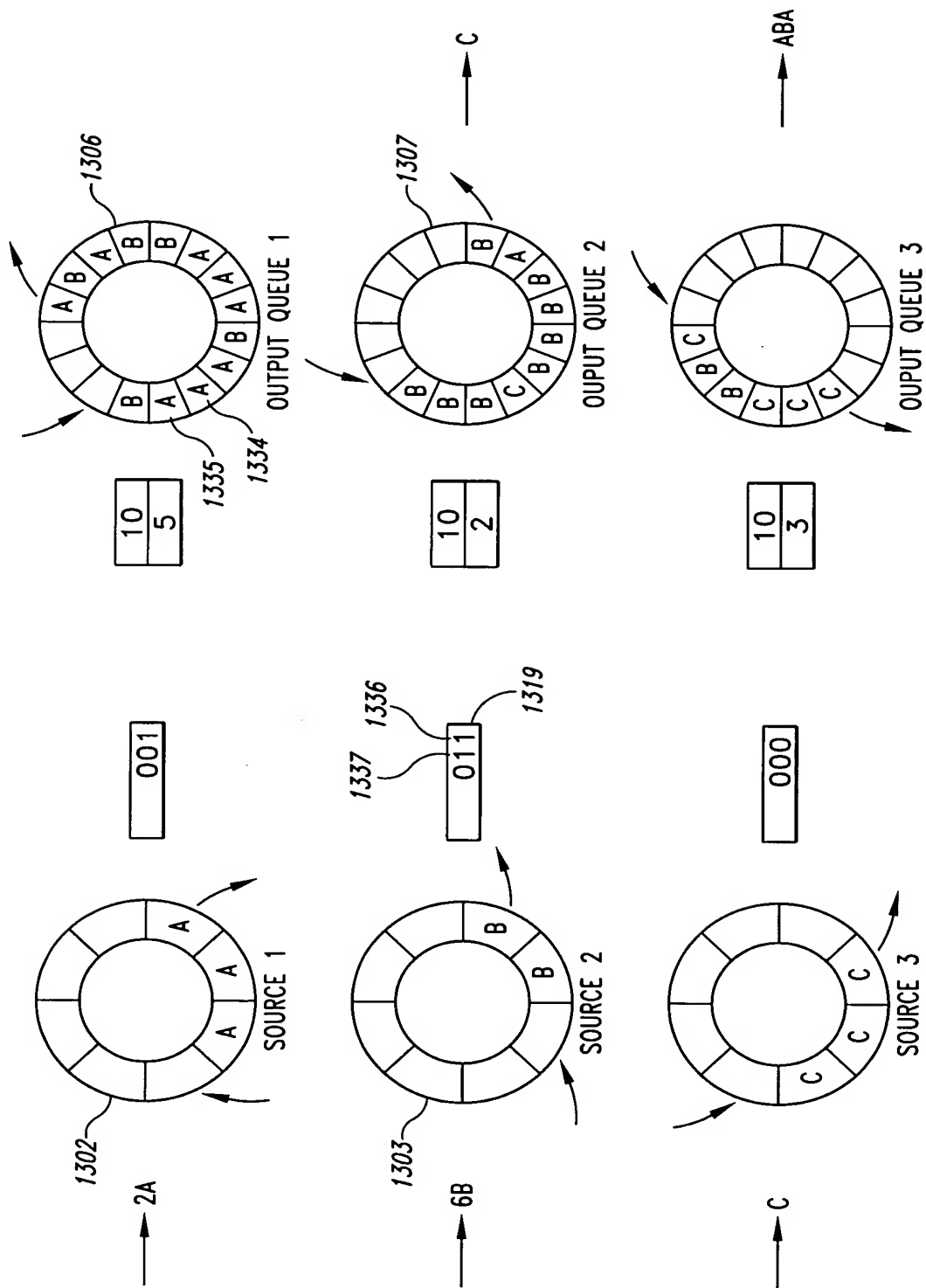


Fig. 13E

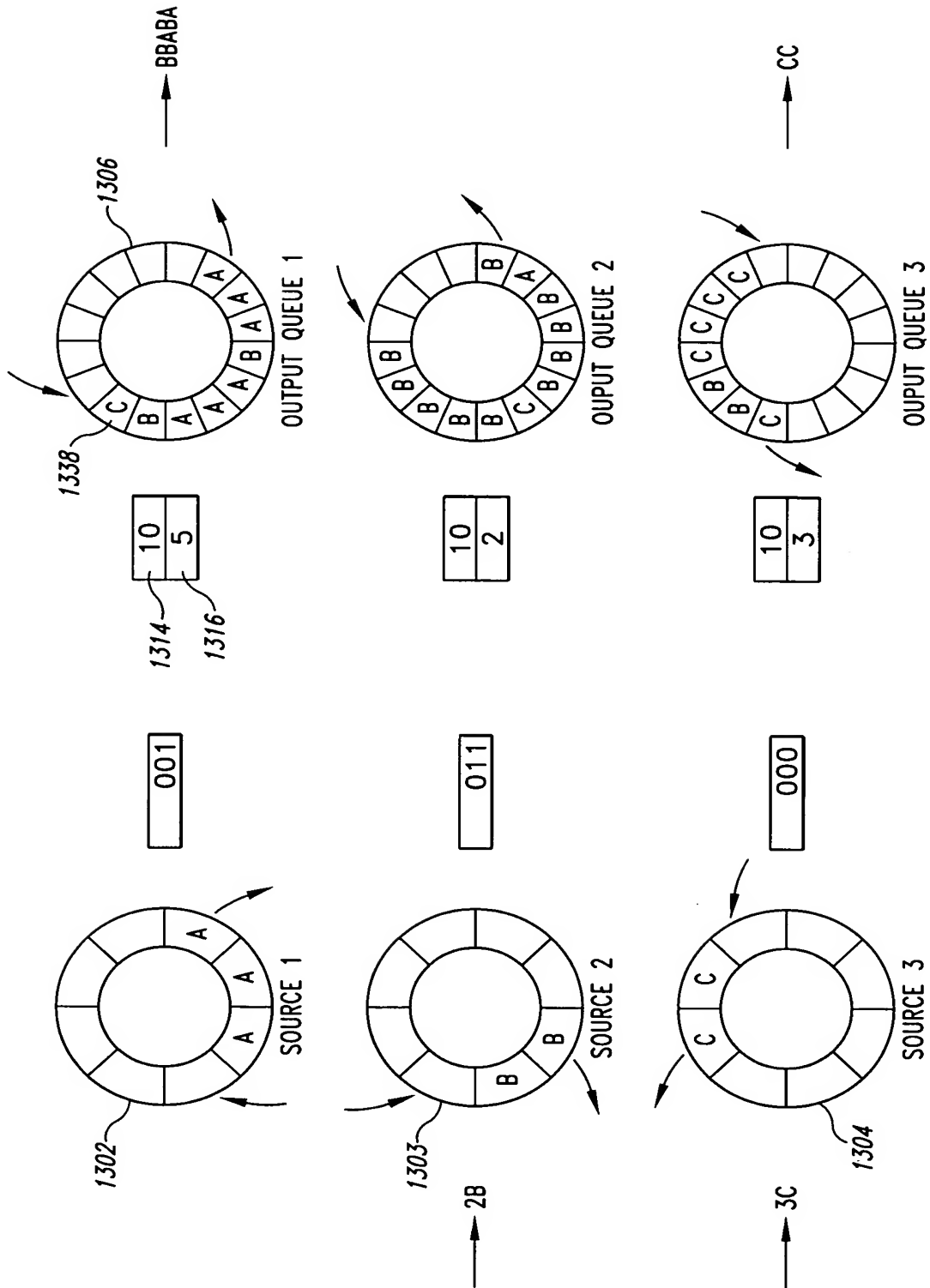


Fig. 13F

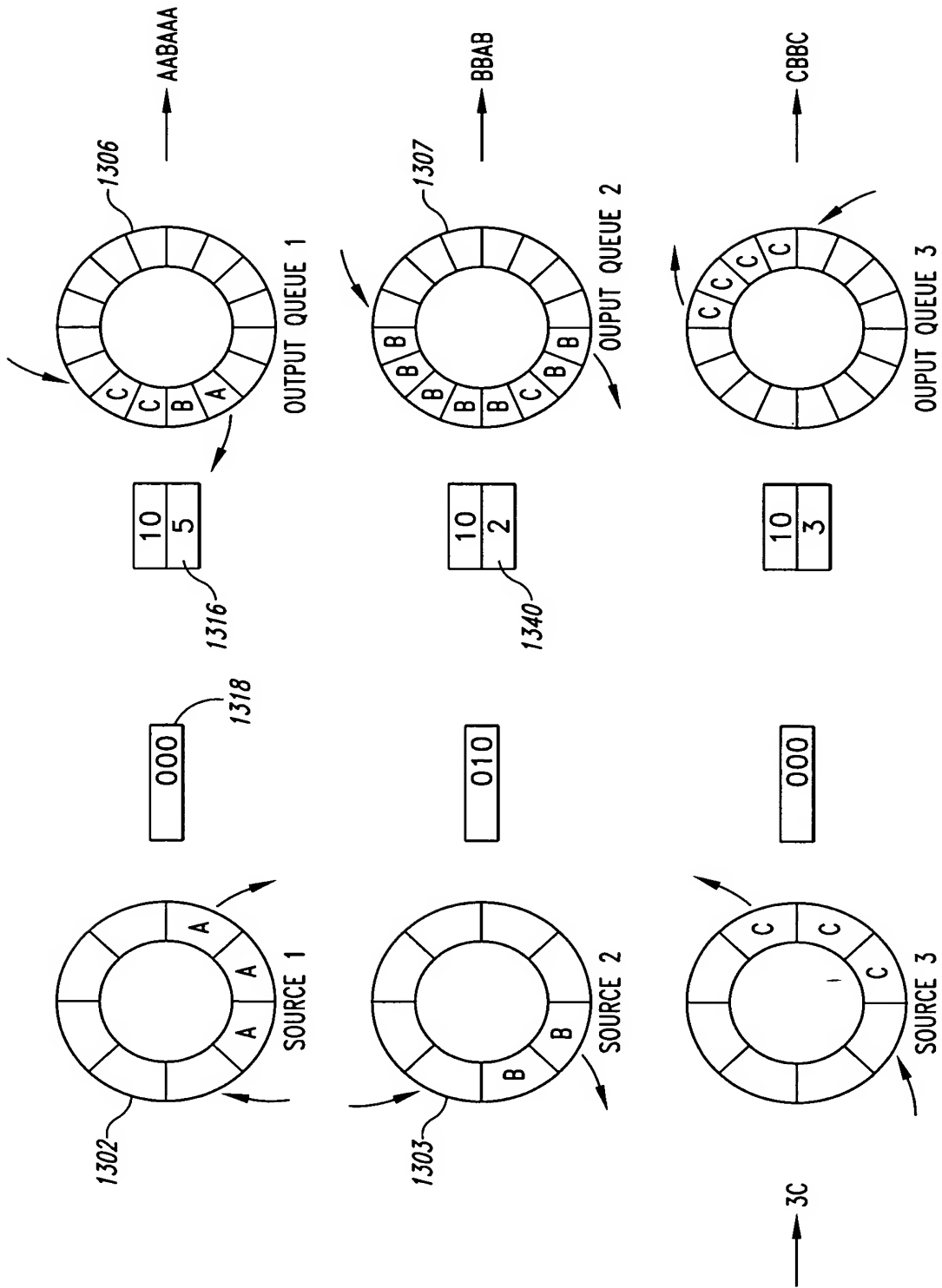


Fig. 13G

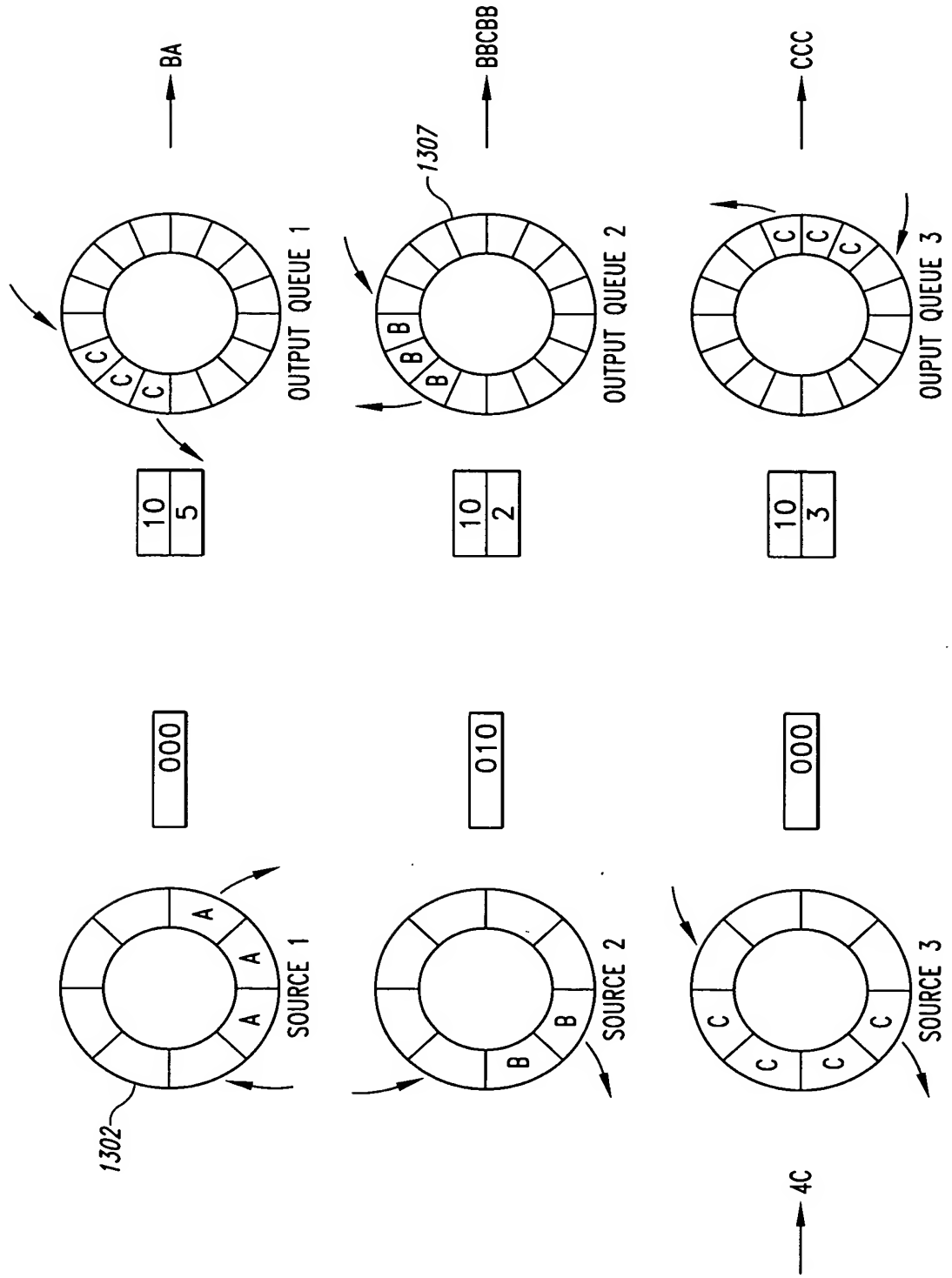


Fig. 13H

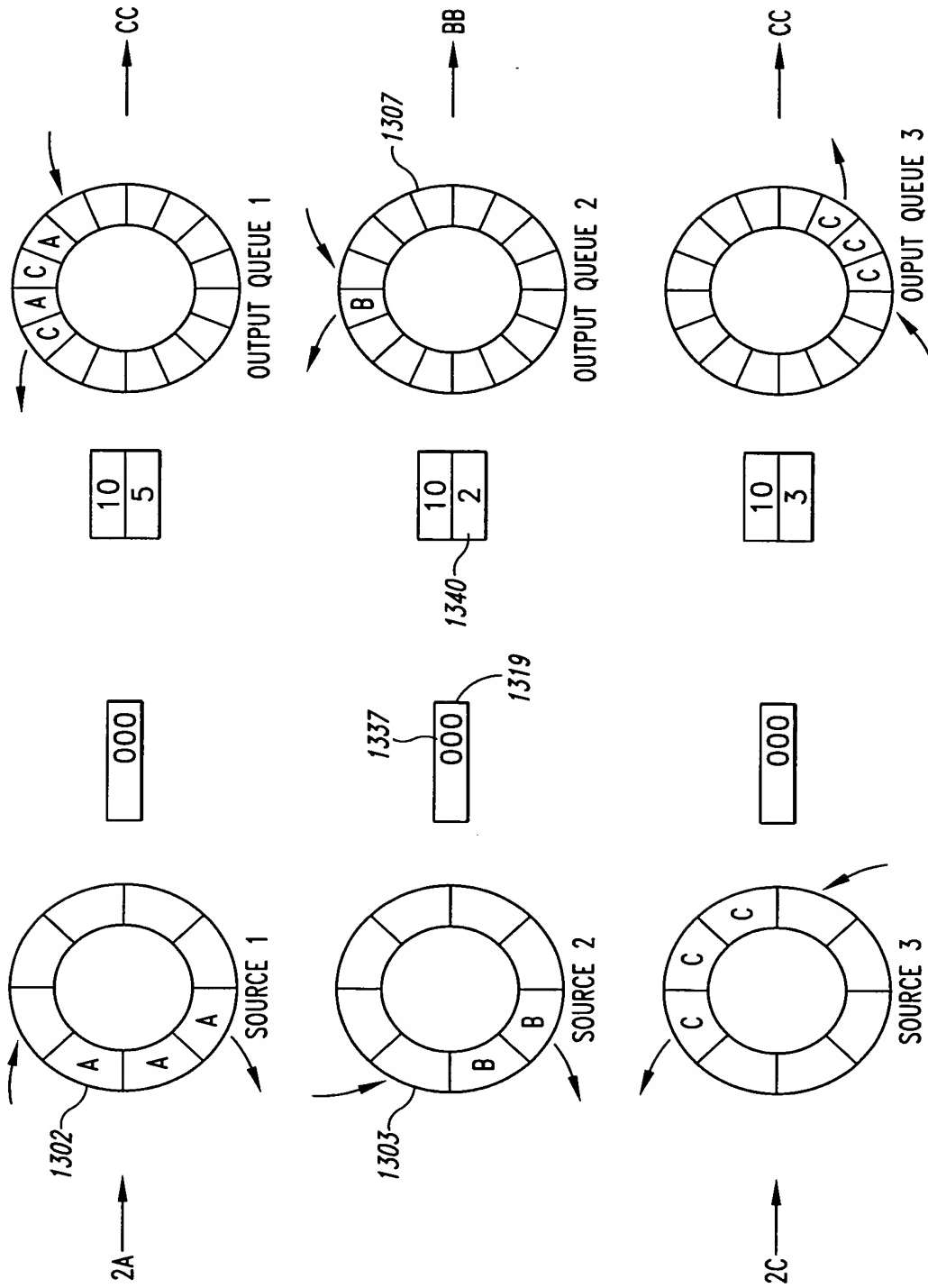


Fig. 13I

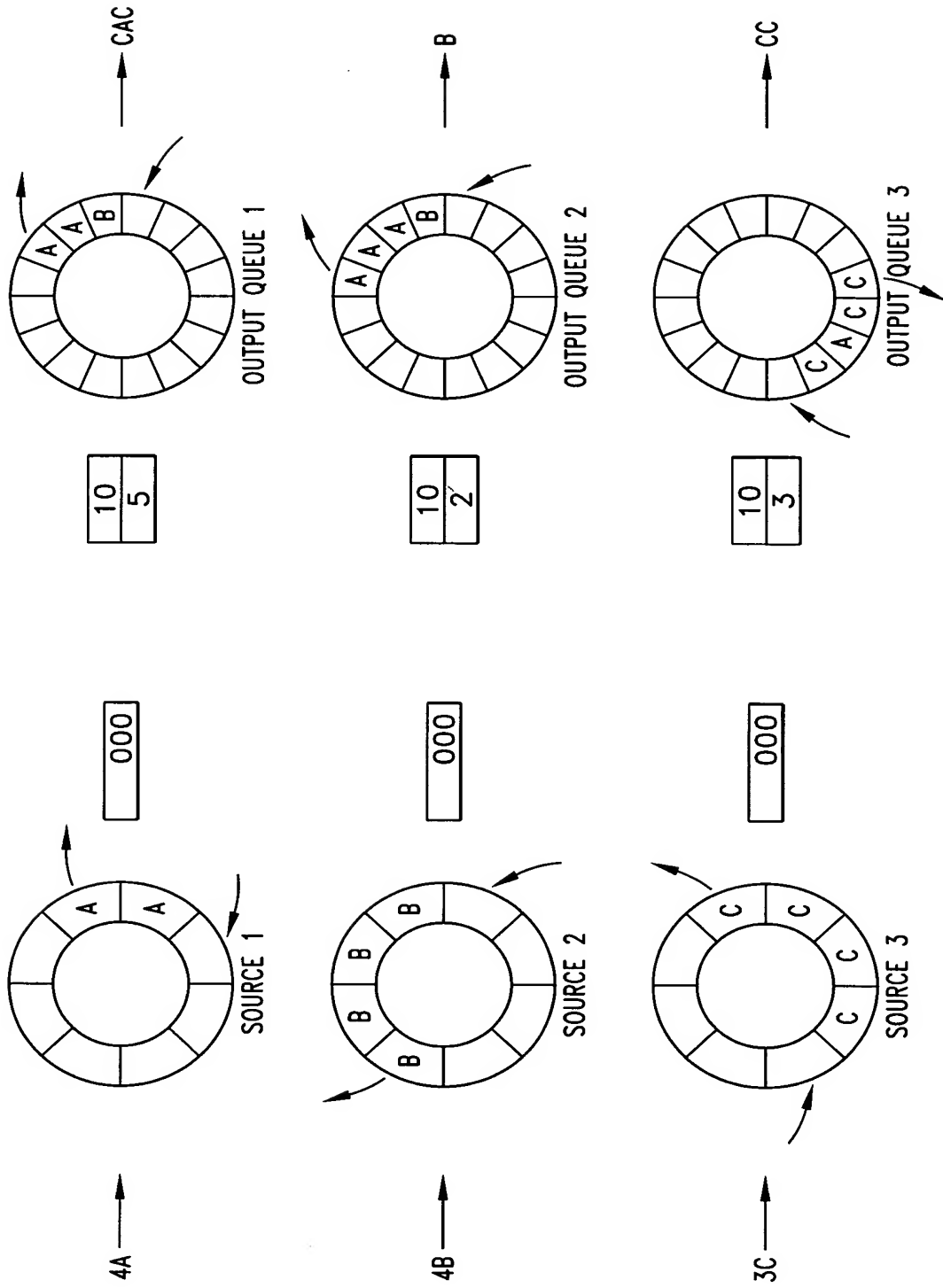


Fig. 13J

```
1 const FORWARD_AFTER_CONTROL=2;
2 typedef void (*callbackFunction)();
3
4 class message
5 {
6     private:
7         char msg[1000];
8         int msgNo;
9     public:
10         int  getLength();
11         int  getSource();
12         int  getDestination();
13         char* getMessage();
14         void setNo (int no);
15         int  getNo();
16         message();
17         ~message();
18 };
19
20 class messageDescriptor
21 {
22     private:
23         message* msgPtr;
24         bool    flag;
25
26     public:
27         message* getMsg();
28         void     setMsg(message *m);
29         void     zeroFlag();
30         bool     getFlag();
31         bool     query();
32         messageDescriptor();
33         ~messageDescriptor();
34 };
35
```

Fig. 14A

```

36 class cQueue
37 {
38     private:
39         messageDescriptor queue[QUEUE_SIZE];
40         int head;
41         int tail;
42         bool inc(int & ptr);
43         bool dec(int & ptr);
44
45     public:
46         bool empty();
47         bool full();
48         int num();
49         bool queueMDesc (messageDescriptor & msg);
50         bool dequeueMDesc (messageDescriptor & msg);
51         cQueue();
52         virtual ~cQueue();
53 };
54
55 class transceiver
56 {
57     public:
58
59         flowControl();
60         releaseFlowControl();
61         kick();
62         startTransceiver(cQueue *out, callbackFunction signalSend,
63                         cQueue *in, callbackFunction signalReceive);
64         transceiver();
65         ~transceiver();
66 };
67
68 class port;
69
70 class portList
71 {
72     public:
73         addPort (port *p);
74         removePort(port *p);
75         port* firstPort();
76         port* nextPort();
77         bool in(port *p);
78         bool empty();
79         bool clear();
80 };
81

```

Fig. 14B

```

82 class portDirectory
83 {
84     public:
85         port *getPortFromSource(int sid);
86 };
87
88 class port
89 {
90     private:
91         cQueue receiveQueue;
92         cQueue transmitQueue;
93         bool overflow;
94         bool flowControlled;
95         int highThreshold;
96         int lowThreshold;
97         portList controlled;
98         portList controlledBy;
99         portDirectory* portDir;
100         transceiver portHardware;
101
102     public:
103         void receiveMessage(messageDescriptor m, port *sPort);
104         void incomingMessage();
105         void outgoingMessage();
106         void receiveFlowControl(port *sPort);
107         void receiveFlowControlRelease(port *sPort);
108         port(portDirectory *p, int low, int high);
109         ~port();
110 };

```

Fig. 14C

25/27

```
1 void port::receiveMessage(messageDescriptor m, port *sPort)
2 {
3     if (transmitQueue.num() > highThreshold - 2)
4     {
5         if (!controlled.in(sPort))
6         {
7             sPort->receiveFlowControl(this);
8             controlled.addPort(sPort);
9         }
10        overflow = TRUE;
11    }
12    if (!transmitQueue.full())
13    {
14        transmitQueue.queueMDesc(m);
15        portHardware.kick();
16    }
17 }
18
19 void port::incomingMessage()
20 {
21     messageDescriptor m;
22     port *p;
23
24     if (controlledBy.empty())
25     {
26         if (flowControlled)
27         {
28             flowControlled = FALSE;
29             portHardware.releaseFlowControl();
30         }
31         while (!receiveQueue.empty() && !flowControlled)
32         {
33             receiveQueue.dequeueMDesc(m);
34             p = portDir->getPortFromSource((m.getMsg()->getSource()));
35             p->receiveMessage(m, p);
36         }
37         portHardware.kick();
38     }
39 }
40
```

Fig. 14D

```

41 void port::outgoingMessage()
42 {
43     port* nxt;
44
45     if (overflow && transmitQueue.num() < lowThreshold)
46     {
47         overflow = FALSE;
48         nxt = controlled.firstPort();
49         while (nxt)
50         {
51             nxt->receiveFlowControlRelease(this);
52             nxt = controlled.nextPort();
53         }
54         controlled.clear();
55     }
56 }
57
58 void port::receiveFlowControl(port *sPort)
59 {
60     int num = FORWARD_AFTER_CONTROL;
61     messageDescriptor m;
62     port *p;
63
64     controlledBy.addPort(sPort);
65     portHardware.flowControl();
66
67     //optional
68     while (num-- && !receiveQueue.empty())
69     {
70         receiveQueue.dequeueMDesc(m);
71         p = portDir->getPortFromSource((m.getMsg())->getSource());
72         p->receiveMessage(m, p);
73     }
74 }
75

```

Fig. 14E

```
76 void port::receiveFlowControlRelease(port *sPort)
77 {
78     controlledBy.removePort(sPort);
79     incomingMessage();
80 }
81 port::port(portDirectory *p, int low, int high)
82 {
83     cQueue* t = &(transmitQueue);
84     cQueue* r = &(receiveQueue);
85     overflow = FALSE;
86     flowControlled = FALSE;
87     highThreshold = high;
88     lowThreshold = low;
89     portDir = p;
90     portHardware.startTransceiver(t, port::outgoingMessage(),
91                                   r, port::incomingMessage());
92 }
```

Fig. 14F